Detecting behavioural anomalies in system logs using log sequence analysis

Sahishnu Snehit Kumar
CSE Department
IIIT Bhubaneswar
Bhubaneswar, India
b523055@iiit-bh.ac.in

Matta Krishna Kumari
CSE Department
SRM University-AP
Amaravati, India
krishnakumari.m@srmap.edu.in

Nikhil Tripathi
CSE Department
IIT (ISM) Dhanbad
Dhanbad, India
nikhiltripathi@iitism.ac.in

***Abstract*—Modern operating systems generate massive volumes of complex log data, making it challenging to promptly detect malicious activities and system anomalies. Traditional security systems are primarily designed to counter high-volume attacks but often fail to identify low-rate threats that blend into normal traffic patterns. Existing anomaly detection methods typically rely on feature-based models and struggle to capture the underlying behavioral patterns reflected in logs. To address this gap, we propose a behavior-based anomaly detection system that leverages log sequence analysis. By transforming log entries into unique log sequences and modeling normal system behavior, the system can identify subtle deviations indicative of malicious activity. Experimental results demonstrate that the proposed approach enables accurate, real-time detection.**

***Index Terms*—Log Anomaly Detection, MITRE ATT&CK, Log Sequence Analysis, Machine Learning**

I. Introduction

Modern operating systems are frequent targets of malware that constantly evolves to bypass traditional security defenses. Attackers often design their methods to closely mimic normal user or system behavior, making timely intrusion detection more challenging. Conventional signature-based Intrusion Detection Systems (IDS) rely on fixed patterns and regular updates, which limit their ability to detect zero-day attacks [1]. To overcome these limitations, many studies have applied Machine Learning (ML) techniques to automate malware detection. However, most existing approaches rely on feature-based models that require carefully engineered features, restricting their scalability and adaptability. These models often fail to generalize across different environments and perform poorly when faced with novel or evasive threats [2], [3]. This gap highlights the need for behavior-based solutions that analyze log sequences to provide more flexible, generalizable, and interpretable detection.

Behavior-based detection methods offer a promising alternative by shifting the focus from static features to the dynamic behavior of processes and system interactions. By analyzing sequences of system logs, these approaches can uncover unusual activity patterns indicative of malicious intent. Recent advances, such as graph-based sequence anomaly detection, have shown strong results in capturing malicious behaviors even when logs contain noise, making them valuable for detecting malware in modern platforms [4]. However, graph-based approaches often suffer from high computational cost, scalability challenges, and limited interpretability, which hinder their practicality for real-time log analysis. Building on this perspective, our work explores a lightweight, behavior-focused strategy to strengthen malware detection in modern operating systems.

In this work, we propose a hybrid framework for real-time anomaly detection based on behavioral log sequence analysis. Our key contributions are:

- We capture a baseline of normal system behavior by collecting logs from our own Windows 11 environment. For malicious activity, we utilize a publicly available malware log dataset aligned with the MITRE ATT&CK framework.
- We adapt behavior-based log sequence analysis to detect deviations from normal activity, enabling the identification of malicious actions.
- We conduct extensive experiments across varying thresholds and window sizes to evaluate detection accuracy, recall, and F1-score, demonstrating robust performance under diverse conditions.

The rest of the paper is organized as follows. Section II reviews related work. Section III details the proposed detection technique, while Section IV presents the experimental results. Finally, Section V concludes the paper and outlines future directions.

II. Related Work

System logs provide a chronological record of logs within computing systems and are extensively used for troubleshooting, performance monitoring, and anomaly detection. In operating systems, logs (Application, Security, and System) contain structured and semi-structured records of user activities, system processes, and error conditions. They are valuable for identifying security breaches, operational failures, and performance issues [5]. However, the massive volume, heterogeneity, and velocity of modern log data make manual inspection infeasible [6], [7].

A. Statistical and Rule-Based Approaches

Early detection methods relied on statistical analysis and threshold-based rules. Dwyer and Truta [5] applied standard-

deviation–based thresholding to detect deviations in log frequencies, enabling alerts for unusual login attempts or abnormal error patterns. While interpretable and easy to implement, such methods were limited to volume anomalies and ignored temporal relationships between logs.

B. Machine Learning-Based Methods

With the growth of large-scale distributed systems, ML has been employed to model normal log patterns and identify deviations. He et al. [6] evaluated six state-of-the-art supervised and unsupervised approaches—including Logistic Regression, Decision Trees, and clustering methods—demonstrating the potential of unsupervised learning when labeled data is unavailable. Zeufack et al. [8] advanced this line of work with a fully unsupervised real-time OPTICS clustering framework, addressing latency limitations of batch detection.

Landauer et al. [9] reviewed widely used public datasets (e.g., HDFS, BGL) and found that they often fail to reflect the complexities of real-world logs, raising concerns about the generalizability of ML-based methods trained on such data.

C. Deep Learning and Representation Learning

Deep learning approaches have been widely explored to capture complex patterns in system logs, particularly the temporal dependencies in sequential data. Sequence-based models such as DeepLog [10] and LogRobust [11] leverage LSTM networks to learn normal execution flows and flag deviations as anomalies. However, these approaches depend heavily on accurate log parsing, which is often error-prone due to evolving formats [12]. To overcome parsing limitations, NeuralLog, proposed by Le and Zhang [12], uses a Transformer-based architecture that processes raw log messages with BERT embeddings, avoiding information loss from template extraction. More recently, Wu et al. [13] conducted a systematic comparison of multiple log representation techniques and demonstrated that the choice of representation, along with aggregation strategies and parsing methods, significantly affects anomaly detection performance. Despite these advances, deep learning methods require large labeled datasets, incur high computational costs, and often act as black boxes, limiting their interpretability and practicality for heterogeneous logs.

D. Motivation for Log Sequence Analysis

Anomalies are often in the form of disruptions in log order, missing steps, or irregular timing rather than as isolated abnormal logs [9]. Sequence analysis preserves these temporal dependencies, enabling detection of patterns that point-wise or distribution-based ML methods typically miss [9]. Practitioners also report that sequence-level approaches yield higher accuracy and interpretability compared to single-log analysis [7].

However, existing ML and deep learning methods show critical drawbacks for log anomaly detection:

- **Weak sequence modeling** – Many methods treat logs as independent points, missing ordering, timing, and dependencies. This prevents detection of multi-step anomalies where individual logs appear benign.
- **Format sensitivity** – Deep models depend on rigid parsing, which fails under log diversity. Sequence-level modeling is more robust, capturing behavior without strict parsing.
- **Limited practicality** – Black-box deep models are costly and hard to interpret. Sequence analysis enables lightweight, low-latency detection with clearer explanations.

By explicitly modeling log ordering, timing, and co-occurrence, log sequence analysis addresses these limitations: it captures behavioral anomalies overlooked by frequency-based methods, maintains robustness across heterogeneous log formats, and provides interpretable results suited for real-time operational use.

III. Proposed Detection Technique

The proposed detection technique uses log sequence analysis to learn the normal behavioral patterns from structured log sequences and identify deviations using lookahead pairs and mismatch scoring [14]. The method operates in two phases: (1) Offline Modeling (Learning) Phase, and (2) Online Detection Phase, which evaluates new log sequences against the learned model to identify anomalies.

A. Learning Phase

This phase builds a database of valid short-range log sequences from the benign lookahead database.

1) Log Normalization and Preprocessing: The dataset captured in our experiments was stored in CSV format and contained the following fields:

- `Timestamp` – date and time when the log occurred.
- `Log ID` – unique numeric identifier for the log type.
- `Level` – severity of the log (*Information*, *Warning*, or *Error*).
- `Source` – application or service that generated the log.
- `Message` – human-readable description containing contextual details such as usernames, file paths, or executed actions.
- `Log Name` – category of the log (*Application*, *Security*, *System*, *Configuration*, or *Forwarded Logs*).

Preprocessing refined the `Message` field by removing volatile components such as timestamps, hexadecimal addresses, and session identifiers, followed by text normalization to eliminate nonsemantic variations. Each refined `Message` was mapped to a unique hash value, ensuring that different instances of the same log type were consistently represented in a canonical form.

2) Log Sequence Construction: Following preprocessing, each hashed message was arranged in chronological order to construct structured log sequences:

$$L_1 \rightarrow L_2 \rightarrow \cdots \rightarrow L_S$$

where L_i denotes the hashed representation of the i^{th} log and S is the sequence length. This transformation produced a consistent, noise-free representation of system activity, forming the basis for transition mapping in subsequent steps.

3) Lookahead Database Construction: The detailed procedure is presented in Algorithm 1. From each log sequence, a database of valid short-range sequences, denoted as $D_{lookahead}$, was generated using a sliding window of size n. For every position in the sequence, all possible log pairs within the lookahead distance $1 \leq k \leq n$ were extracted in the form $(L_{\text{current}}, L_{\text{future}}, k)$ **(Steps 2–12)**. These lookahead pairs capture localized behavioral patterns in normal system operation. The complete set of unique pairs forms the reference database used during anomaly detection to identify deviations from established benign patterns.

Algorithm 1 From Unique Log Sequence to Lookahead Database

Require: $normal\_sequences$: List of normal log sequences
Require: n: Lookahead window size
Ensure: $D_{lookahead}$: Set of normal lookahead pairs
1: Initialize an empty set $D_{lookahead}$
2: **for all** $sequence \in normal\_sequences$ **do**
3: Initialize an empty set $sequence\_pairs$
4: **for** $i \leftarrow 0$ to $\text{len}(sequence) - 1$ **do**
5: **for** $k \leftarrow 1$ to n **do**
6: **if** $i + k < \text{len}(sequence)$ **then**
7: $pair \leftarrow (sequence[i], sequence[i + k], k)$
8: Add $pair$ to $sequence\_pairs$
9: **end if**
10: **end for**
11: **end for**
12: Add all pairs from $sequence\_pairs$ to $D_{lookahead}$
13: **end for**
14: **return** $D_{lookahead}$

B. Detection Phase

In the detection phase, incoming log sequences are evaluated against the database of benign lookahead pairs constructed during the learning phase to determine whether they represent normal or anomalous behavior. The complete detection process is detailed in Algorithm 2.

1) Sequence Preprocessing and Pair Extraction: Incoming log data undergo the same preprocessing steps as in the learning phase: volatile elements in the `Message` field are removed, the text is normalized, and each message is mapped to its unique hash value. The resulting hashed sequence is then processed using the same sliding window size n applied during training, extracting all possible lookahead pairs in the form $(L_{\text{current}}, L_{\text{future}}, k)$ **(Steps 1–6)**.

2) Detection Using Lookahead Log Sequences: Following preprocessing, both normal and malicious datasets are transformed into *lookahead pairs*, which capture short-range logs transitions within a sliding window of size n. A database of valid lookahead pairs, $D_{lookahead}$, is constructed from the normal dataset. During detection, incoming log sequences lookahead is compared against this database, and any lookahead pair not present in $D_{\text{lookahead}}$ is counted as a mismatch **(Steps 7–13)**.

Algorithm 2 Anomaly Detection Using Lookahead Pairs

Require: $test\_sequence$: Sequence of logs to be evaluated
Require: $D_{lookahead}$: Database of normal lookahead pairs
Require: n: Lookahead window size
Require: t: Detection threshold
Ensure: Classification as Normal or Anomalous
1: $L \leftarrow \text{len}(test\_sequence)$
2: $mismatches \leftarrow 0$
3: **for** $i \leftarrow 0$ to $L - 1$ **do**
4: **for** $k \leftarrow 1$ to n **do**
5: **if** $i + k < S$ **then**
6: $pair \leftarrow (test\_sequence[i], test\_sequence[i + k], k)$
7: **if** $pair \notin lookahead\_db$ **then**
8: $mismatches \leftarrow mismatches + 1$
9: **end if**
10: **end if**
11: **end for**
12: **end for**
13: Compute $MismatchScore$ using Equation 1
14: **if** $MismatchScore >$ t **then**
15: **return** Anomalous
16: **else**
17: **return** Normal
18: **end if**

To account for variations in sequence length, the anomaly indicator—termed the *MismatchScore*—is defined as:

$$\textit{MismatchScore} = \frac{\text{Number of Mismatches}}{n \times \left(\frac{S-n+1}{2}\right)} \tag{1}$$

where S denotes the total number of logs in the sequence.

The normalized $MismatchScore$ is compared against a predefined threshold t. If the score exceeds t, the sequence is classified as anomalous; otherwise, it is classified as normal **(Steps 14–17)**.

IV. Experiment & Results

This section presents the experimental study conducted to assess the performance of the proposed detection approach. The experiments were conducted on a Windows 11 workstation equipped with an Intel® Core™ i7 processor and 16 GB RAM. The lookahead window size n was varied between 1 and 5 to study its impact on detection performance, and the mismatch score threshold t was tuned empirically based on validation results from the normal dataset.

1) Normal Log Collection for Training: Normal logs were collected from a Windows 11 system by capturing routine user activities via the Windows Command Prompt. This process generated over 100000 log entries covering a wide range of benign system behaviors and including diverse `Log ID`'s, `Source`'s, and severity `Level`'s. The logs were preprocessed following the procedure described in Section III-A1 to remove volatile components and ensure consistent log representation. From the refined dataset, unique logs were extracted and

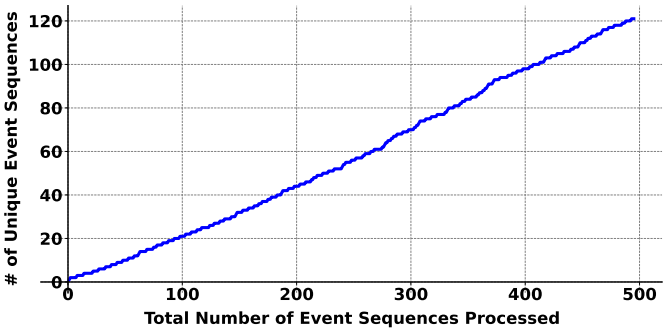

Fig. 1: Unique log sequences

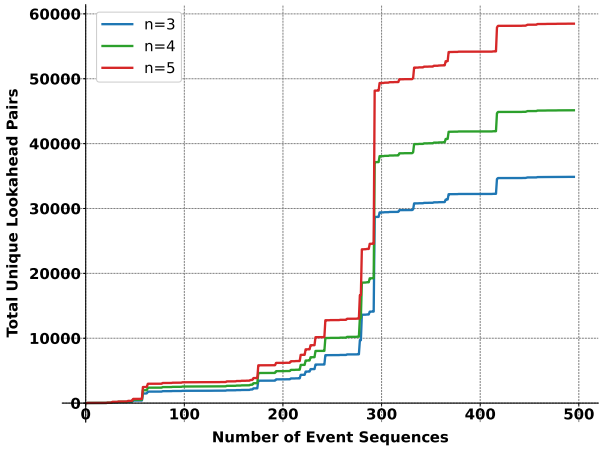

Fig. 2: Lookahead pairs for various window sizes

organised into valid lookahead pairs as described in Section III-A3, forming the lookahead database of permissible short-range log sequences used for model training. Figures 1 and 2 illustrate the trends observed with the growth in benign Windows log entries and lookahead pairs. It can be observed from Figure 1 that the unique log sequence count increases continuously as the number of benign log entries increases. Thus, it is virtually infeasible to determine the exact amount of benign Windows log traces required to capture the complete normal behavior of the system [15], [16]. On the other hand, Figure 2 shows the number of lookahead pairs with respect to different log sequences. It is evident that the number of lookahead pairs increases as the log sequences grow, up to nearly 400 sequences. Beyond that point, almost no new lookahead pairs are found. This suggests that the lookahead pairs database can capture nearly complete benign characteristics of Windows logs at that stage. The study also concludes that the lookahead pairs database is more effective in capturing the expected behavior of Windows logs than a unique log sequence database.

2) Malicious Log Collection for Testing: The malicious dataset, comprising more than 4200 log entries, was obtained from an open-source repository on GitHub. These logs were generated during controlled execution of malware samples mapped to relevant ***MITRE ATT&CK*** techniques (e.g., `T1059` – Command and Scripting Interpreter, `T1105` – Ingress Tool Transfer) and include behaviors such as process creation anomalies, unauthorised access attempts, and suspicious system modifications. After applying the same preprocessing and normalisation steps as for the benign dataset, the logs were sequenced and transformed into lookahead pairs. The mismatch score of each testing entry was then computed using Equation 1, by comparing its lookahead pairs against the benign lookahead database. Figure 3 shows the Cumulative Distribution Function (CDF) of mismatch values with a window size of five. Benign sequences remain tightly clustered near zero, with almost all values below 0.2, whereas malicious sequences span the full range (0.0–1.0), with a significant proportion above 0.1.

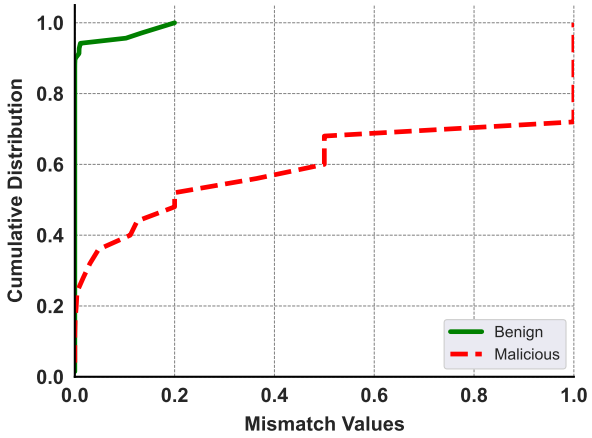

Fig. 3: Mismatch score of benign and malicious sequences

A comparative analysis of lookahead pairs further reinforces this separation: benign activity is dominated by repetitive, stable patterns (e.g., `Information`→`Information`), while malicious traces frequently display irregular transitions such as `Error`→`Error` or abrupt shifts between unrelated log types. The malicious dataset also contained a higher proportion of rare or previously unseen patterns, underscoring significant deviations from normal behavior. These distinctions form the foundation of the proposed detection method, which models legitimate activity from benign lookahead pairs and flags anomalous deviations during real-time monitoring.

A. Evaluation Metrics

The effectiveness of the proposed method is evaluated using four performance metrics:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{3}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{4}$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5}$$

Here, the symbols TP, FP, TN, and FN are defined as:

TABLE I: Performance at varying thresholds ($n = 3$)

Threshold (t)	Accuracy	Precision	Recall	F1-Score
0.01	89.47%	82.61%	76.00%	79.17%
0.05	96.84%	100%	88.00%	93.62%
0.1	91.57%	100%	68.00%	80.95%
0.15	95.78%	95.65%	88.00%	91.66%
0.2	89.47%	100%	60.00%	74.99%

TABLE II: Performance at different window sizes ($t = 0.05$)

Window Size (n)	Accuracy	Precision	Recall	F1-Score
3	96.84%	100%	88.00%	93.62%
5	96.84%	100%	88.00%	91.67%
7	95.79%	95.65%	88.00%	91.67%

- **True Positive (TP)**: Anomalous sequences that are correctly classified as anomalies.
- **False Positive (FP)**: Normal sequences that are incorrectly flagged as anomalies.
- **True Negative (TN)**: Normal sequences that are correctly classified as normal.
- **False Negative (FN)**: Anomalous sequences that are incorrectly classified as normal.

B. Results and Analysis

Table I shows the effect of varying the *MismatchScore* threshold t with window size $n = 3$. At very low thresholds ($t = 0.01$), recall is higher (76.00%) but precision drops (82.61%) because of a high number of false positives. Increasing the threshold to $t = 0.05$ yields the best overall performance, achieving **96.84% accuracy**, **100% precision**, 88.00% recall, and a 93.62% F1-Score. Beyond $t = 0.1$, recall decreases because of an increase in the number of false negatives. Thus, $t = 0.05$ offers the best trade-off in our experiments.

We also varied the window size ($n = 3, 5, 7$) while fixing $t = 0.05$. As shown in Table II, the method remained robust across window sizes, with only minor performance differences. Smaller windows ($n = 3$) performed slightly better and are more efficient, making them preferable for real-time detection.

Finally, Table III compares our model with supervised and unsupervised baselines. While unsupervised methods achieve moderate recall, they suffer from poor precision because of high false alarms. A supervised classifier improves accuracy but still lacks precision. In contrast, our log sequence model achieves the best overall balance, delivering **high accuracy (96.84%) and zero false positives (100% precision)** while maintaining strong recall.

C. Complexity Analysis

Lookahead pair extraction time: We experimented with varying the window size to measure its impact on the time required for extracting lookahead pairs from Windows log sequences. The results of this experiment are shown in Figure 4.

We observe that the extraction time grows almost linearly with increasing window size, with larger windows significantly increasing computational cost. This highlights the trade-off between richer context (large w) and timely detection, making smaller to medium window sizes preferable for practical deployment.

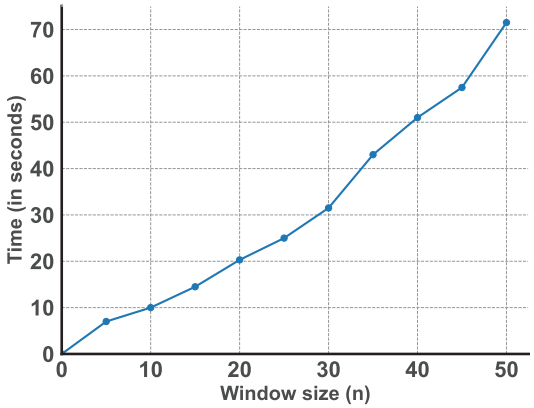

Fig. 4: Execution time based on different window sizes

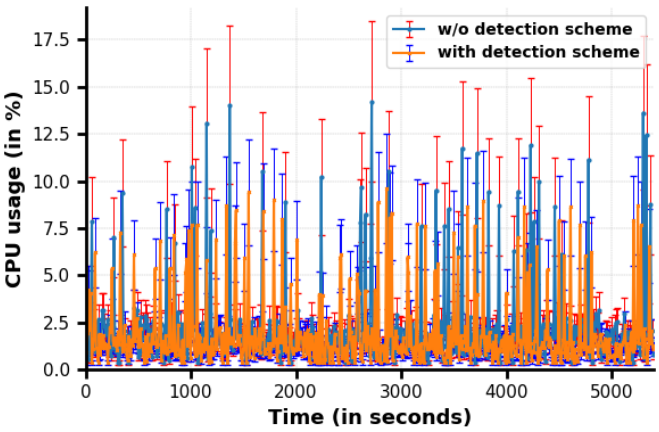

Fig. 5: CPU usage

Processing overhead: We deployed the detection approach on a Windows 11 machine with a 12th Gen Intel® Core9-12900F processor and 64 GB RAM. CPU and memory usage were monitored using the `psutil` library [17]. Figure 5 shows CPU utilization with and without the detection scheme enabled. We find that CPU consumption remains comparable in both cases, with slightly lower usage when the detection module is active. Similarly, RAM usage increased initially but stabilized at roughly double the idle baseline, still well within system capacity. These results confirm that the proposed framework introduces minimal computational overhead, making it practical for real-time malware detection in Windows 11 environments.

V. Conclusion

The modern operating systems produce rich logs of messages that record the system's behavioral dynamics. We have demonstrated that it is possible to model such logs based on unique message transitions and lookahead pair analysis

TABLE III: Comparison with baseline models

Category	Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Unsupervised	Autoencoder	57.64	18.07	88.79	30.02
	Isolation Forest	10.18	10.18	**99.41**	18.47
	K-Means	31.73	11.99	89.38	21.14
	Ensemble Method	68.87	18.94	62.24	29.04
Supervised	Supervised Classifier	90.60	52.50	85.90	65.16
Log Sequence	Proposed Model	**96.84**	**100.00**	88.00	**93.62**

to build useful anomaly detectors. In controlled experimental conditions, the benign activity had consistent and predictable modes of operation, whereas injected malicious behavior brought irregular shifts, which were precisely identified by our method with high accuracy and high recall at the various thresholds. Although the current dataset is based on a clear distinction between normal and anomalous sequences, real-life applications can be faced with a higher variability, which needs adaptive profile construction to support different settings. In general, this paper shows that behavior-based log sequence analysis has the potential to improve security monitoring system reliability and resiliency.

ACKNOWLEDGMENT

This work is supported by IIT (ISM) Dhanbad under FRS, Project No. MISC 0179.

REFERENCES

[1] K. Berlin, D. Slater, and J. Saxe, "Malicious Behavior Detection using Windows Audit Logs," *arXiv preprint arXiv:1506.04200*, 2015.

[2] E. Raff and C. Nicholas, "A Survey of Machine Learning Methods and Challenges for Windows Malware Classification," *arXiv preprint arXiv:2006.09271*, 2020.

[3] Y. Xie, H. Zhang, and M. A. Babar, "LogGD: Detecting Anomalies from System Logs by Graph Neural Networks," *arXiv preprint arXiv:2209.07869*, 2022.

[4] B. Dong, Z. Chen, H. Wang, L.-A. Tang, K. Zhang, Y. Lin, Z. Li, and H. Chen, "Efficient discovery of abnormal event sequences in enterprise security systems," in *ACM Conference on Information and Knowledge Management*, 2017, pp. 707–715.

[5] J. Dwyer and T. M. Truta, "Finding anomalies in windows event logs using standard deviation," in *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2013, pp. 563–570.

[6] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[7] X. Ma, Y. Li, J. Keung, X. Yu, H. Zou, Z. Yang, F. Sarro, and E. T. Barr, "Practitioners' expectations on log anomaly detection," *IEEE Transactions on Software Engineering*, 2025.

[8] V. Zeufack, D. Kim, D. Seo, and A. Lee, "An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis," *High-Confidence Computing*, vol. 1, no. 2, p. 100030, 2021.

[9] M. Landauer, F. Skopik, and M. Wurzenberger, "A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1354–1375, 2024.

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[11] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 807–817.

[12] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[13] X. Wu, H. Li, and F. Khomh, "On the effectiveness of log representation for log-based anomaly detection," *Empirical Software Engineering*, vol. 28, no. 6, p. 137, 2023.

[14] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *IEEE symposium on security and privacy*. IEEE, 1996, pp. 120–128.

[15] N. Tripathi, "Delays have dangerous ends: Slow http/2 dos attacks into the wild and their real-time detection using event sequence analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 3, pp. 1244–1256, 2023.

[16] M. Swain, N. Tripathi, and K. Sethi, "Identifying communication sequence anomalies to detect DoS attacks against MQTT," *Computers & Security*, p. 104526, 2025.

[17] "Rodola g, psutil 5.9.8," https://pypi.org/project/psutil/, 2024, accessed: 13.04.2024.